Data Visualization in R

Kevin Johnson

October 30, 2014

1 Introduction

Ggplot2 is widely considered the best R package for visualizing data. It has a unique way of implementing plots so it can have a steep learning curve, but once you get it the syntax is very powerful.

The core function used to start every plot is `ggplot()`. The next most important function is `aes()` which stands for aesthetics. Plots are built by adding layers onto the base function that specify what you want the plot to show (you literally use the + operator to add layers). Parameters passed to the `aes()` function are passed on to every layer in the plot.

2 Scatterplots

We'll start with something easy, a simple scatterplot. Ggplot2 comes with several datasets preloaded, one of which is called `mtcars`. It has data on cars extracted from the 1974 Motor Trend US magazine. The variables included are:

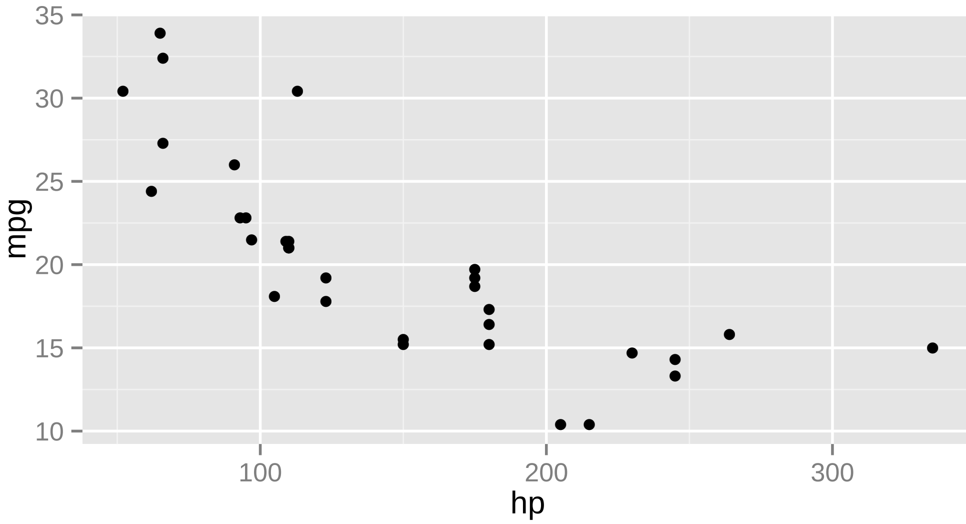
- mpg: Miles/(US) gallon
- cyl: Number of cylinders
- disp: Displacement (cu.in.)

- hp: Gross horsepower
- drat: Rear axle ratio
- wt: Weight (lb/1000)
- qsec: 1/4 mile time
- vs: V/S
- am: Transmission (0 = automatic, 1 = manual)
- gear: Number of forward gears
- carb: Number of carburetors

```
head(mtcars)

##                    mpg cyl disp  hp drat    wt  qsec vs am gear carb
## Mazda RX4         21.0   6  160 110 3.90 2.620 16.46  0  1    4    4
## Mazda RX4 Wag     21.0   6  160 110 3.90 2.875 17.02  0  1    4    4
## Datsun 710        22.8   4  108  93 3.85 2.320 18.61  1  1    4    1
## Hornet 4 Drive    21.4   6  258 110 3.08 3.215 19.44  1  0    3    1
## Hornet Sportabout 18.7   8  360 175 3.15 3.440 17.02  0  0    3    2
## Valiant           18.1   6  225 105 2.76 3.460 20.22  1  0    3    1
```

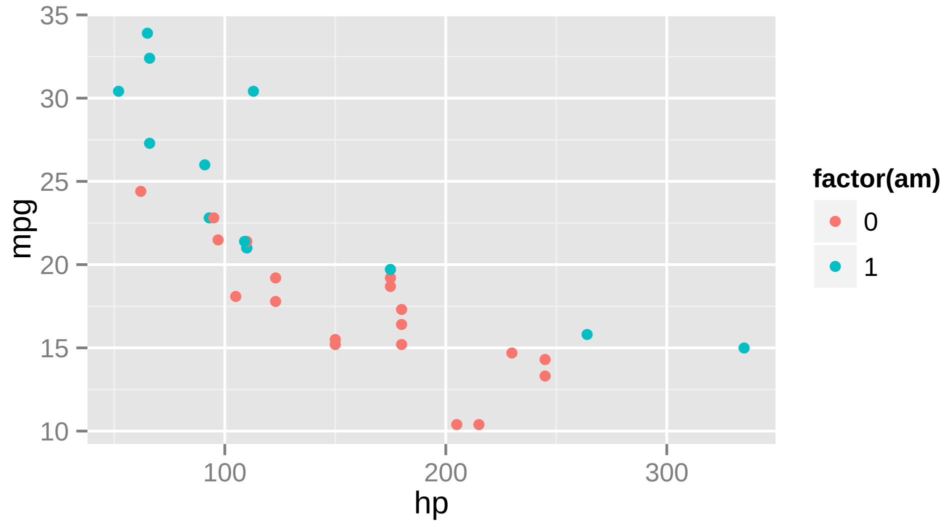
The `geom_point()` function takes in x and y values and creates a scatterplot of the points. Let's plot the horsepower of the car on the x-axis and the miles per gallon on the y-axis.

```
ggplot(data = mtcars, aes(x = hp, y = mpg)) +
    geom_point()
```



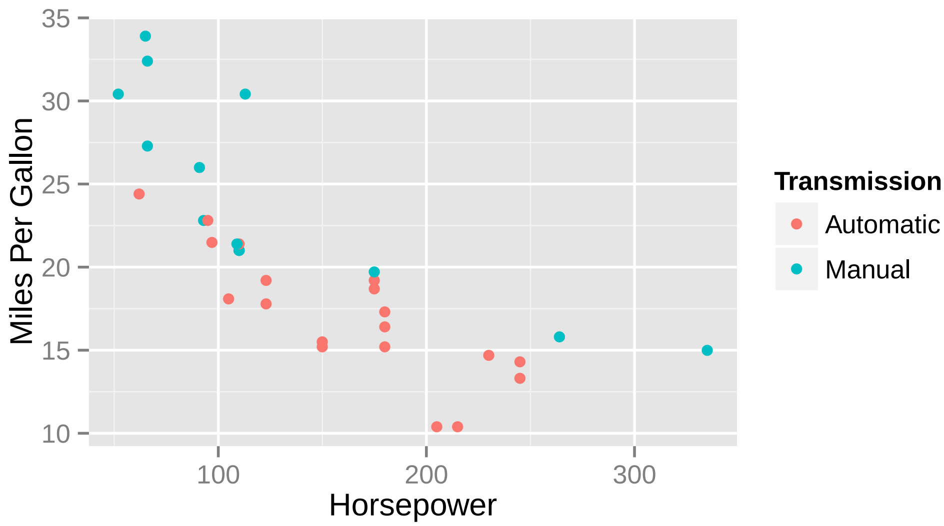
The geom_point() function can take several parameters that determine what the points will look like. You can control things like color, transparency, size, and shape. One of the best features of ggplot2 is that you can set easily set these aesthetic properties to be based on variables in your dataset. For example, let's say we want to have the points be different colors based on if the car is manual or automatic (stored in am).

```
ggplot(data = mtcars, aes(x = hp, y = mpg, color = factor(am))) +
    geom_point()
```



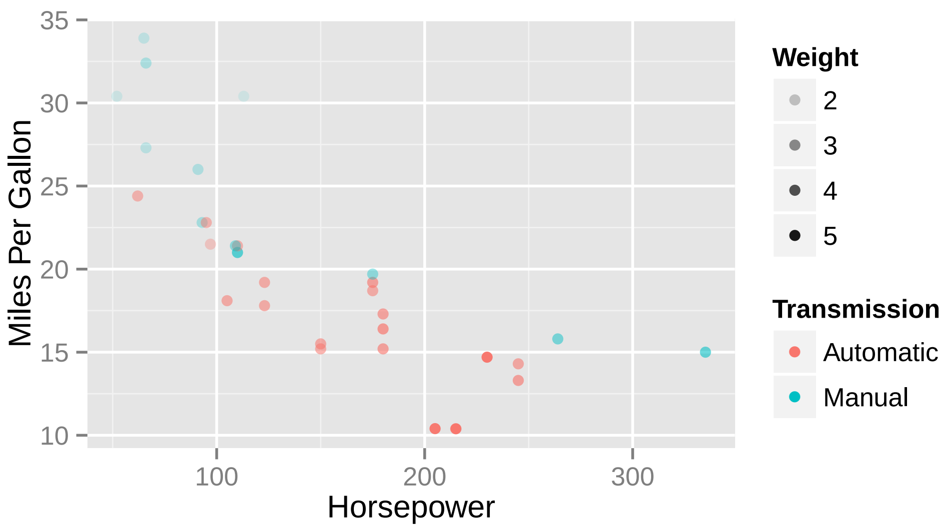
I'll talk about themeing graphs in more detail later, but for now let's just look at how to handle legends and scales in ggplot2. Obviously we don't want our legend to be titled "factor(am)", and we would like to have more informative labels for the colors and each axis. We can accomplish both of theses goals using `scale_color_discrete()` and `labs()`.

```
ggplot(data = mtcars, aes(x = hp, y = mpg,
    color = factor(am))) +
    geom_point() +
    scale_color_discrete(labels = c("Automatic", "Manual")) +
    labs(color = "Transmission", x = "Horsepower",
        y = "Miles Per Gallon")
```



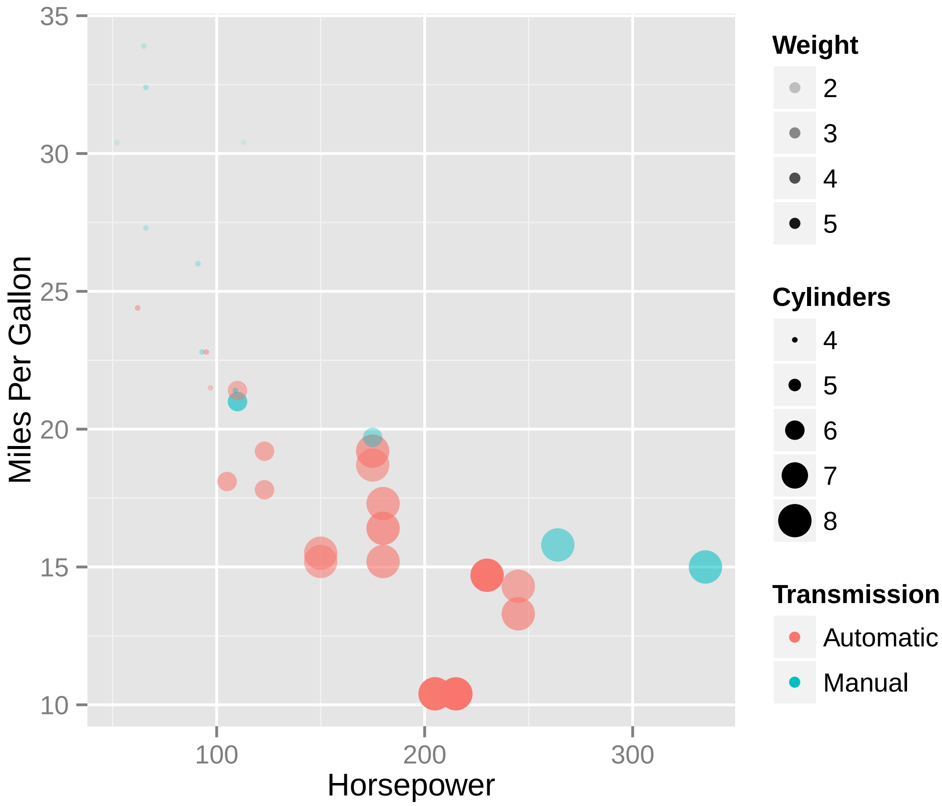
What if we want the opacity of each point to be proportional to the weight of the vehicle? The `alpha` parameter lets us control the transparency of a given layer.

```
ggplot(data = mtcars, aes(x = hp, y = mpg,
    color = factor(am), alpha = wt)) +
    geom_point() +
    scale_color_discrete(labels = c("Automatic", "Manual")) +
    labs(color = "Transmission", x = "Horsepower",
        y = "Miles Per Gallon", alpha = "Weight")
```



What about making the size of the point proportional to the number of cylinders in the vehicle?

```
ggplot(data = mtcars, aes(x = hp, y = mpg,
    color = factor(am), alpha = wt,
    size = cyl)) +
    geom_point() +
    scale_color_discrete(labels = c("Automatic", "Manual")) +
    labs(color = "Transmission", x = "Horsepower",
        y = "Miles Per Gallon", alpha = "Weight",
        size = "Cylinders")
```



Of course, this is getting a bit ridiculous now, but the point is that you can do all sorts of interesting things with ggplot by passing parameters through the layers via the `aes()` function.

3 Bar Charts

The `geom_bar()` function allows you to create all sorts of bar charts. We're going to use a different dataset now that comes with ggplot2 called `diamonds`. This dataset contanis prices and other attributes of more than 50,000 diamonds. The variables included are:

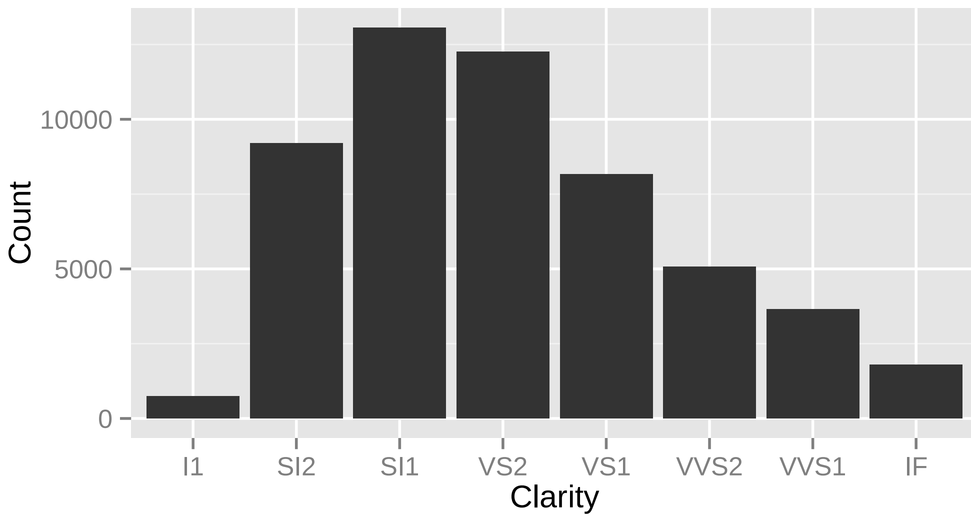
- price: price in US dollars ($326–$18,823)

- carat: weight of the diamond (0.2–5.01)
- cut: quality of the cut (Fair, Good, Very Good, Premium, Ideal)
- colour: diamond colour, from J (worst) to D (best)
- clarity: a measurement of how clear the diamond is (I1 (worst), SI1, SI2, VS1, VS2, VVS1, VVS2, IF (best))
- x: length in mm (0–10.74)
- y: width in mm (0–58.9)
- z: depth in mm (0–31.8)
- depth: total depth percentage
- table: width of top of diamond relative to widest point (43–95)

```
head(diamonds)

##   carat       cut color clarity depth table price    x    y    z
## 1  0.23     Ideal     E     SI2  61.5    55   326 3.95 3.98 2.43
## 2  0.21   Premium     E     SI1  59.8    61   326 3.89 3.84 2.31
## 3  0.23      Good     E     VS1  56.9    65   327 4.05 4.07 2.31
## 4  0.29   Premium     I     VS2  62.4    58   334 4.20 4.23 2.63
## 5  0.31      Good     J     SI2  63.3    58   335 4.34 4.35 2.75
## 6  0.24 Very Good     J    VVS2  62.8    57   336 3.94 3.96 2.48
```

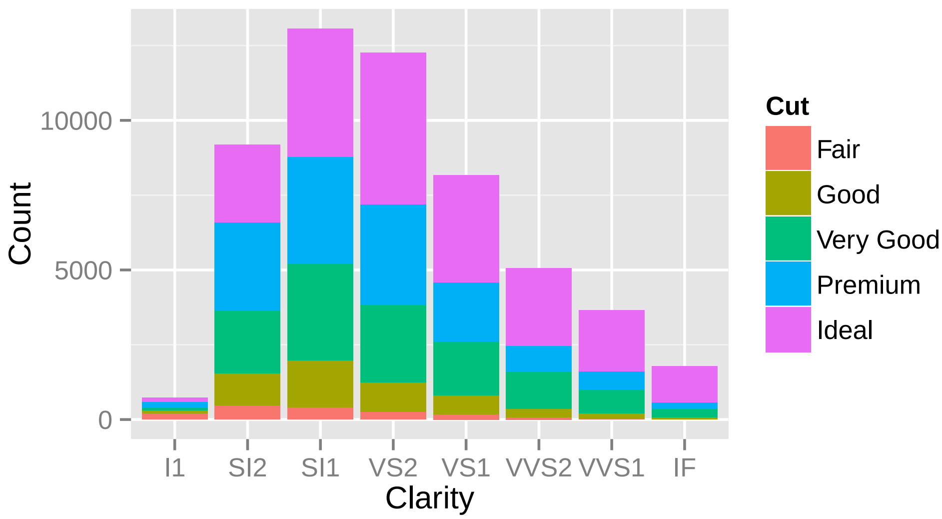
Let's say we want to look at how many diamonds there for each value of clarity. This time we don't need to include a y parameter because the `geom_bar()` function will use the number of data points in each category as the y value. If you want to pass your own y values then you will need to add `stat = "identity"` as a parameter in the `geom_bar()` function.

```
ggplot(data = diamonds, aes(x = clarity)) +
    geom_bar() +
    labs(x = "Clarity", y = "Count")
```



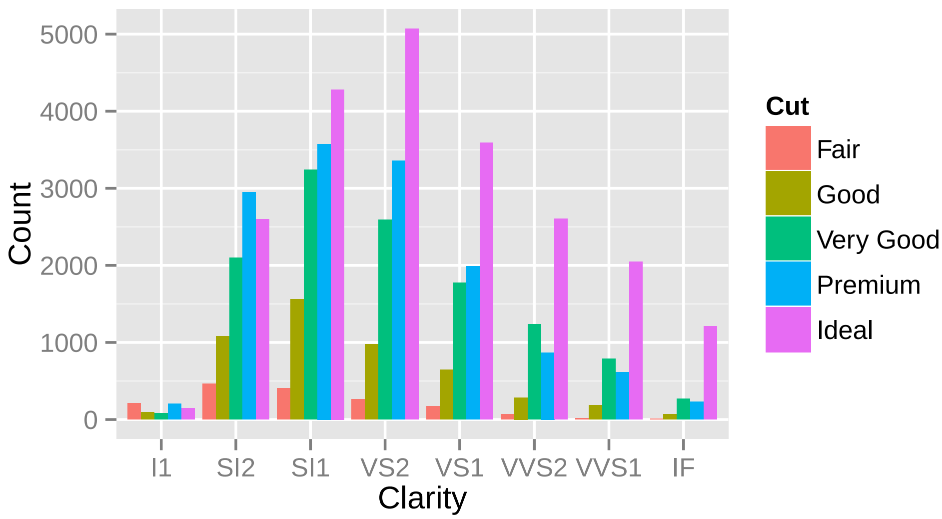
Next, let's color the bars according to the quality of the cut. The `fill` parameter controls the color of the inside of the bar, and the `color` parameter controls the color of the outline of the bar. This distinction is used throughout ggplot.

```
ggplot(data = diamonds, aes(x = clarity, fill = cut)) +
    geom_bar() +
    labs(x = "Clarity", y = "Count", fill = "Cut")
```



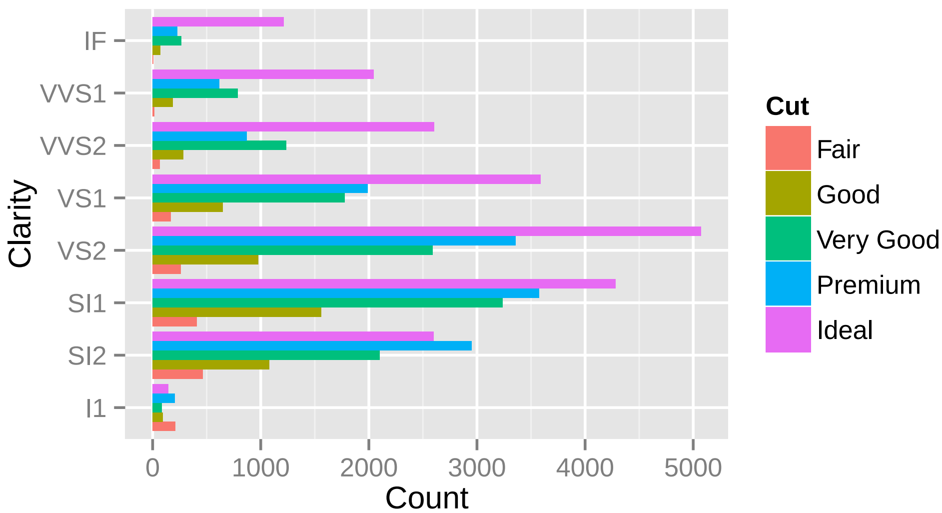
What if we want the bars to be next to each other instead of stacked on top of each other? The `position` parameter allows us to do that by taking in values of `stack`, `dodge`, or `fill`.

```
ggplot(data = diamonds, aes(x = clarity, fill = cut)) +
    geom_bar(position = "dodge") +
    labs(x = "Clarity", y = "Count", fill = "Cut")
```



Often you will come across text labels for your x-axis that are too large to fit. You can rotate your x-axis labels to make them fit (that's an exercise left to the reader), or you can use `coord_flip()` to flip the axes.

```
ggplot(data = diamonds, aes(x = clarity, fill = cut)) +
    geom_bar(position = "dodge") +
    labs(x = "Clarity", y = "Count", fill = "Cut") +
    coord_flip()
```



4 Box Plots

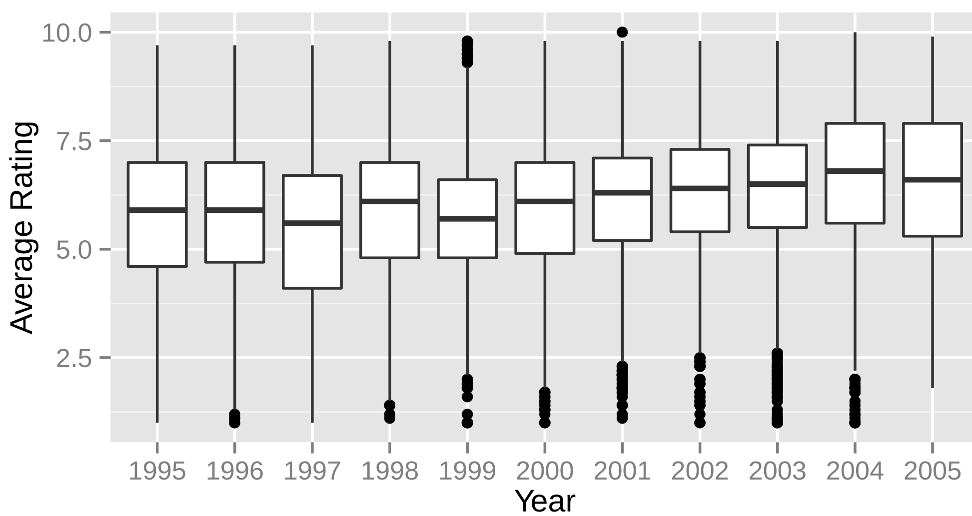
Let's move on to a new dataset called `movies`. This data comes from IMDB and includes 30000 rows with the following variables:

- title: Title of the movie.
- year: Year of release.
- budget: Total budget (if known) in US dollars
- length: Length in minutes.
- rating: Average IMDB user rating.
- votes: Number of IMDB users who rated this movie.
- r1-10: Multiplying by ten gives percentile (to nearest 10%) of users who rated this movie a 1.
- mpaa: MPAA rating.

- action, animation, comedy, drama, documentary, romance, short: Binary variables representing if movie was classified as belonging to that genre.

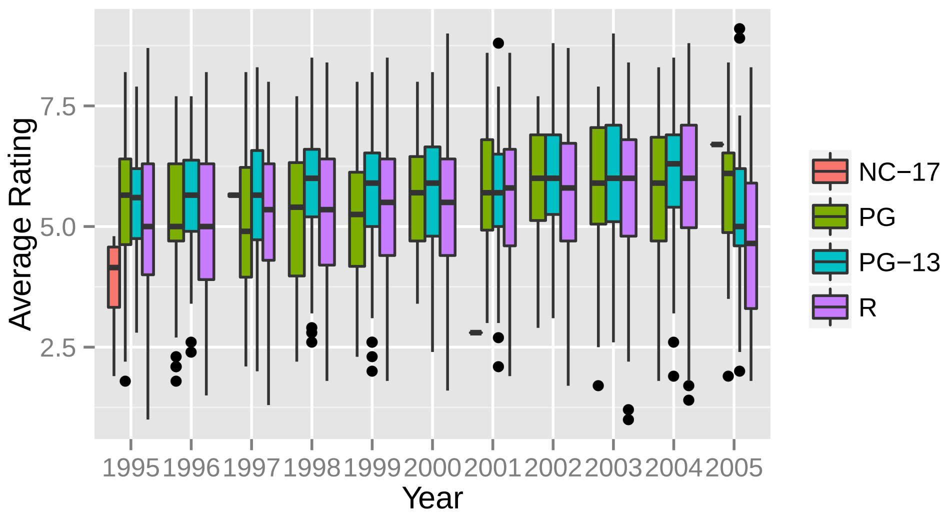
Let's make a boxplot of the average ratings from 1995 to 2005. The `geom_boxplot()` function allows us to do that.

```
ggplot(data = movies[movies$year >= 1995,] ,
    aes(x = factor(year), y = rating)) +
    geom_boxplot() +
    labs(x = "Year", y = "Average Rating")
```



We can also separate this by the MPAA rating. I'm going to remove the legend title because I feel like it is self explanatory in this context.

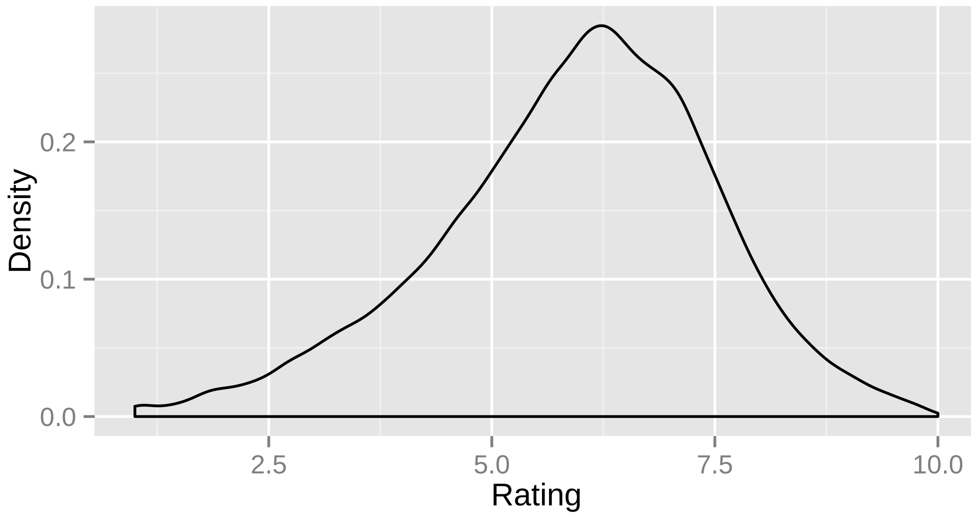
```
ggplot(data = movies[movies$year >= 1995 & movies$mpaa != "",],
    aes(x = factor(year), y = rating, fill = mpaa)) +
    geom_boxplot() +
    labs(x = "Year", y = "Average Rating", fill = "")
```



5 Density

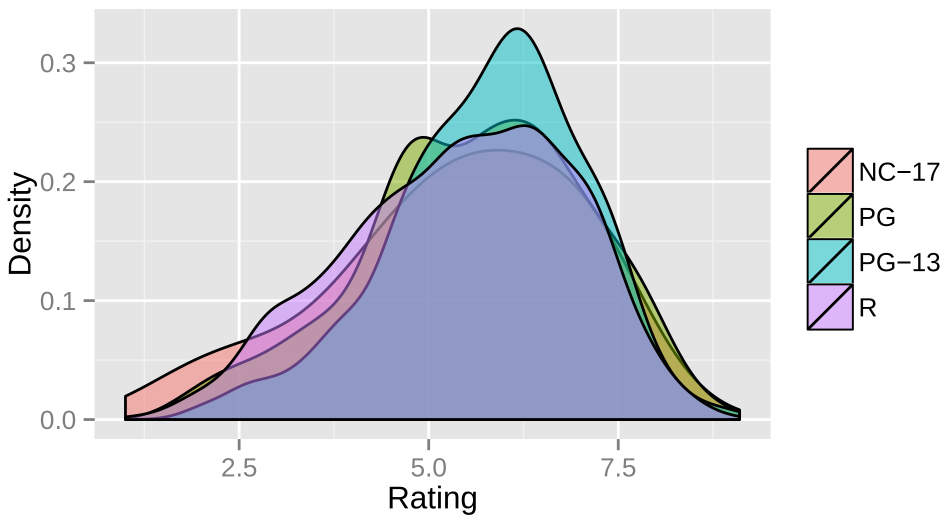
The `geom_density()` function gives a 1d kernel estimate of whatever variable you give it. The function calls the base R `density` function which takes a number of parameters. The default is a Gaussian kernel. Let's look at the distribution of movie ratings.

```
ggplot(data = movies, aes(x = rating)) +
    geom_density() +
    labs(x = "Rating", y = "Density")
```



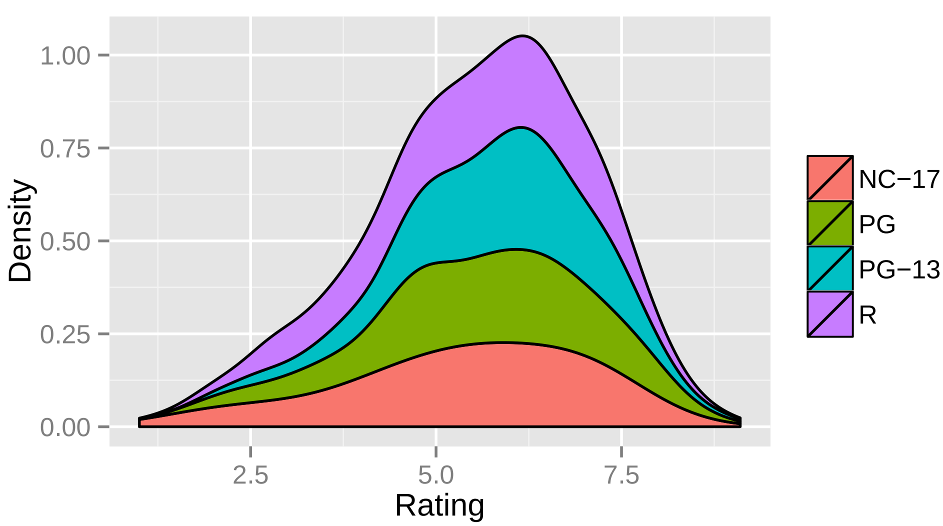
We can separate these densities by MPAA rating. Note that I am lowering the transparency to make it possible to see multiple distributions at once.

```
ggplot(data = movies[movies$mpaa != "",],
    aes(x = rating, fill = mpaa)) +
    geom_density(alpha = 0.5) +
    labs(x = "Rating", y = "Density", fill = "")
```



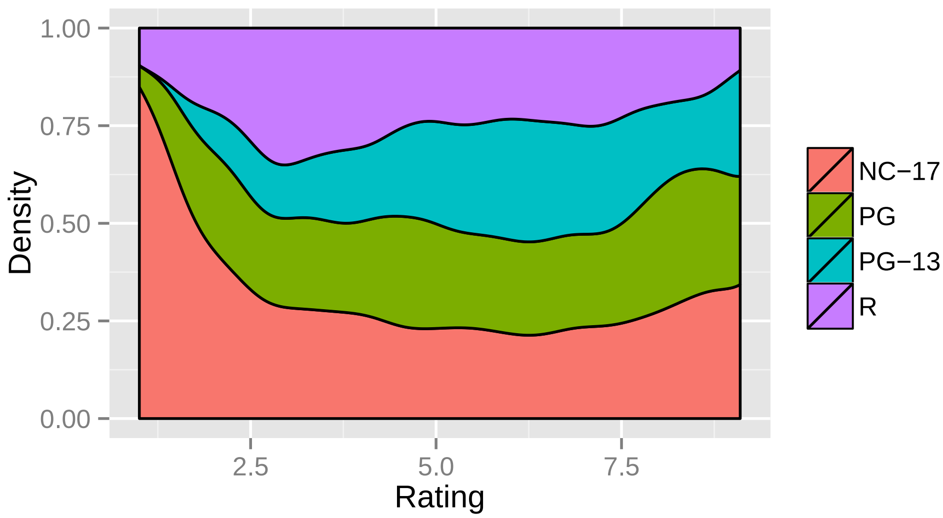
You can also stack these densities on top of each other.

```
ggplot(data = movies[movies$mpaa != "",],
    aes(x = rating, fill = mpaa)) +
    geom_density(position = "stack") +
    labs(x = "Rating", y = "Density", fill = "")
```



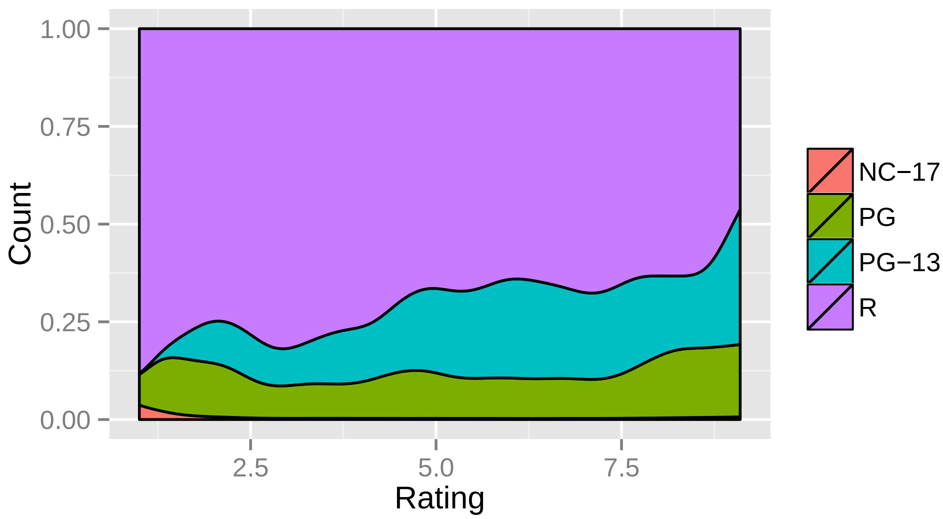
Using `position = "fill"` can be particularly interesting in this context. That will produce a conditional density estimate for each average rating.

```
ggplot(data = movies[movies$mpaa != "",],
    aes(x = rating, fill = mpaa)) +
    geom_density(position = "fill") +
    labs(x = "Rating", y = "Density", fill = "")
```



Changing it to produce a raw count of records rather than a density estimate might be a better to get an actual idea of how many movies of different MPAA ratings are given a particular average IMDB rating. The special `..count..` expression will do that for us.

```
ggplot(data = movies[movies$mpaa != "",],
    aes(x = rating, y = ..count.., fill = mpaa)) +
    geom_density(position = "fill") +
    labs(x = "Rating", y = "Count", fill = "")
```



6 Smoothing

Sometimes you want to be able to visualize a trend, whether it be a simple linear model or a more complicated smoothing model. The `geom_smooth()` function takes in several parameters, the most important of which is `method` which can take the following values:

- `lm`: linear model
- `glm`: generalized linear model
- `gam`: generalized additive model
- `loess`: LOcal regrESSion (locally weighted scatterplot smoothing)
- `rlm`: robust linear model

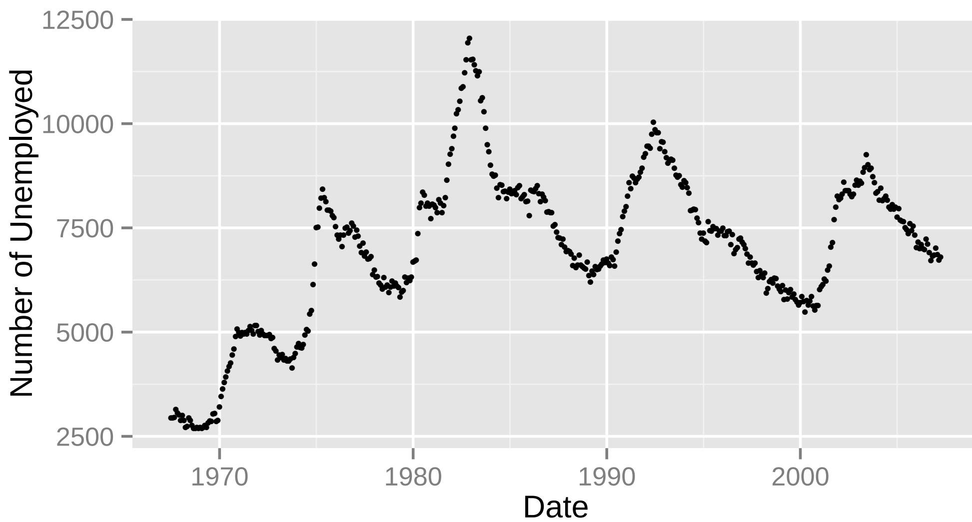
All of these are extensively documented in the R help files. The function defaults to `loess` for data with <1000 samples and defaults to `gam` otherwise. For visualization purposes, leaving it to be default will usually work well enough, but this function is flexible enough to implement any smoothing method you can think of.

Let's look at yet another new dataset called `economics` which includes time series data on US economic metrics. The variables included are:

- date. Month of data collection
- psavert, personal savings rate, `http://research.stlouisfed.org/fred2/series/PSAVERT/`
- pce, personal consumption expenditures, in billions of dollars, `http://research.stlouisfed.org/fred2/series/PCE`
- unemploy, number of unemployed in thousands, `http://research.stlouisfed.org/fred2/series/UNEMPLOY`
- uempmed, median duration of unemployment, in week, `http://research.stlouisfed.org/fred2/series/UEMPMED`
- pop, total population, in thousands, `http://research.stlouisfed.org/fred2/series/POP`

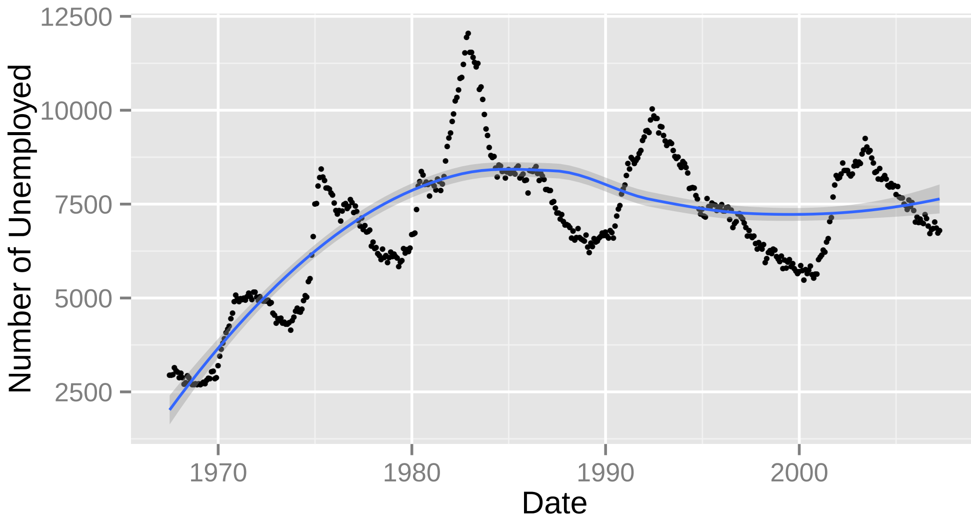
Let's start with something easy like total number of unemployed people over time. First, we have to talk about how dates and times are handled in R. For dates, R has a special variable type called `Date`. These can be created with the `as.Date()` function, but luckily for us our dates are already in the correct format. Ggplot has been coded to handle R date and time objects in a way that makes sense, but custom options can be specified using the `scale_x_time()` function and similar functions.

```
ggplot(data = economics, aes(x = date, y = unemploy)) +
    geom_point(size = 1) +
    labs(x = "Date", y = "Number of Unemployed")
```



Notice how ggplot automatically formats the x-axis in a way that makes sense. Isn't that neat? Next, we'll add a smoothing function to the data to get a better look at what the overall trend is.

```
ggplot(data = economics, aes(x = date, y = unemploy)) +
    geom_point(size = 1) +
    geom_smooth(method = "loess") +
    labs(x = "Date", y = "Number of Unemployed")
```



This is mostly just a proof of concept, in practice you would want to use an actual time series analysis technique for this application (e.g. exponential smoothing, ARIMA).

7 Faceting

What if you want to plot 4 different time series at once, possibly with different scales? The concept of facetting in ggplot allows you to do that. The first step is to transform your data from wide to long format. Wide format is what most datasets look like because it is what makes the most sense to us. Due to the way R handles dataframes, it is often easier to work with long format data from a coding standpoint. Here's what our data currently looks like:

```
head(economics)

##          date   pce    pop psavert uempmed unemploy
## 1 1967-06-30 507.8 198712     9.8     4.5     2944
## 2 1967-07-31 510.9 198911     9.8     4.7     2945
## 3 1967-08-31 516.7 199113     9.0     4.6     2958
```

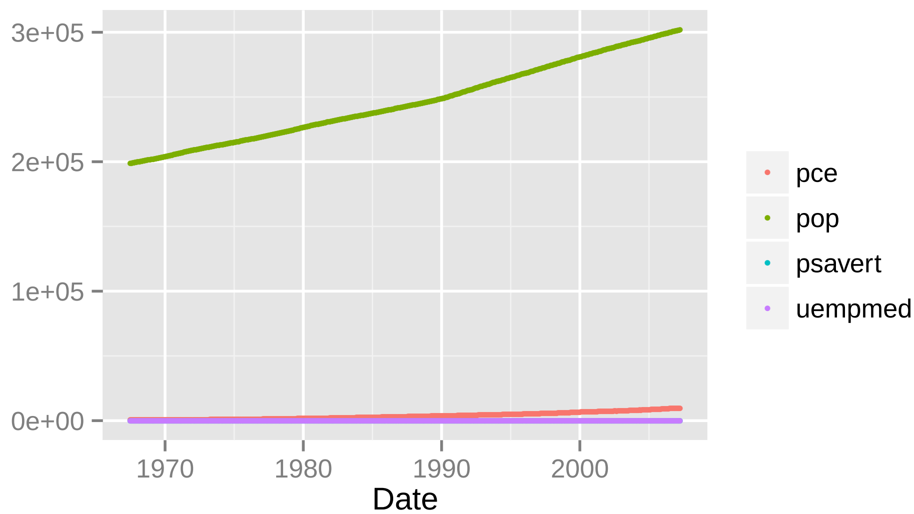
```
## 4 1967-09-30 513.3 199311     9.8     4.9     3143
## 5 1967-10-31 518.5 199498     9.7     4.7     3066
## 6 1967-11-30 526.2 199657     9.4     4.8     3018
```

Now we're going to use the `melt()` function in the `reshape2` package to transform this into long form data, using `date` as an ID variable.

```
library(reshape2)
economicsLong <- melt(economics[,1:5], id = "date")
head(economicsLong)

##         date variable value
## 1 1967-06-30      pce 507.8
## 2 1967-07-31      pce 510.9
## 3 1967-08-31      pce 516.7
## 4 1967-09-30      pce 513.3
## 5 1967-10-31      pce 518.5
## 6 1967-11-30      pce 526.2

tail(economicsLong)

##            date variable value
## 1907 2006-10-31  uempmed   8.2
## 1908 2006-11-30  uempmed   7.3
## 1909 2006-12-31  uempmed   8.1
## 1910 2007-01-31  uempmed   8.1
## 1911 2007-02-28  uempmed   8.5
## 1912 2007-03-31  uempmed   8.7
```

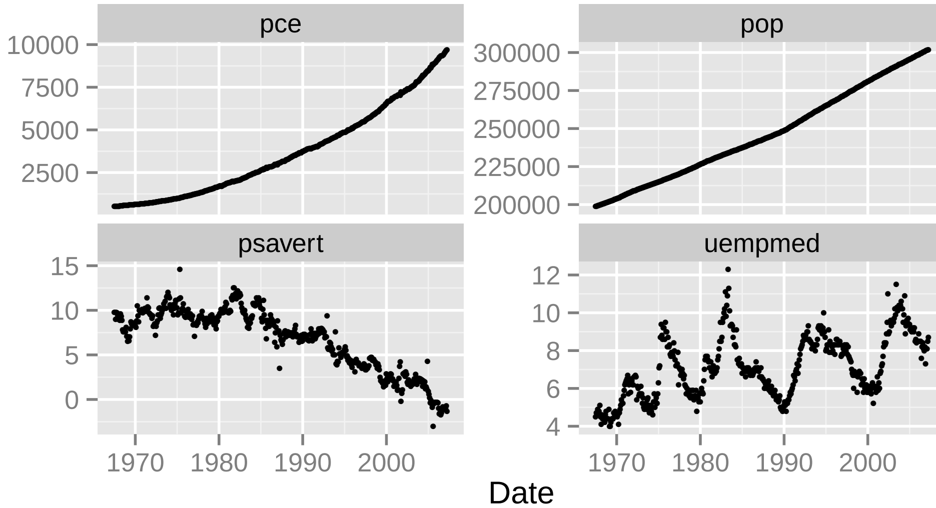
Basically, we have collapsed all of our columns (except `date`) into a single column named `value`, and we keep track of which row corresponds to which of our original variables in a new column called `variable`. Let's look at how we can use this new format to easily create more complicated plots. First, let's plot each variable as a time series on the same plot.

```
ggplot(data = economicsLong, aes(x = date, y = value, color = variable)) +
    geom_point(size = 1) +
    labs(x = "Date", y = "", color = "")
```



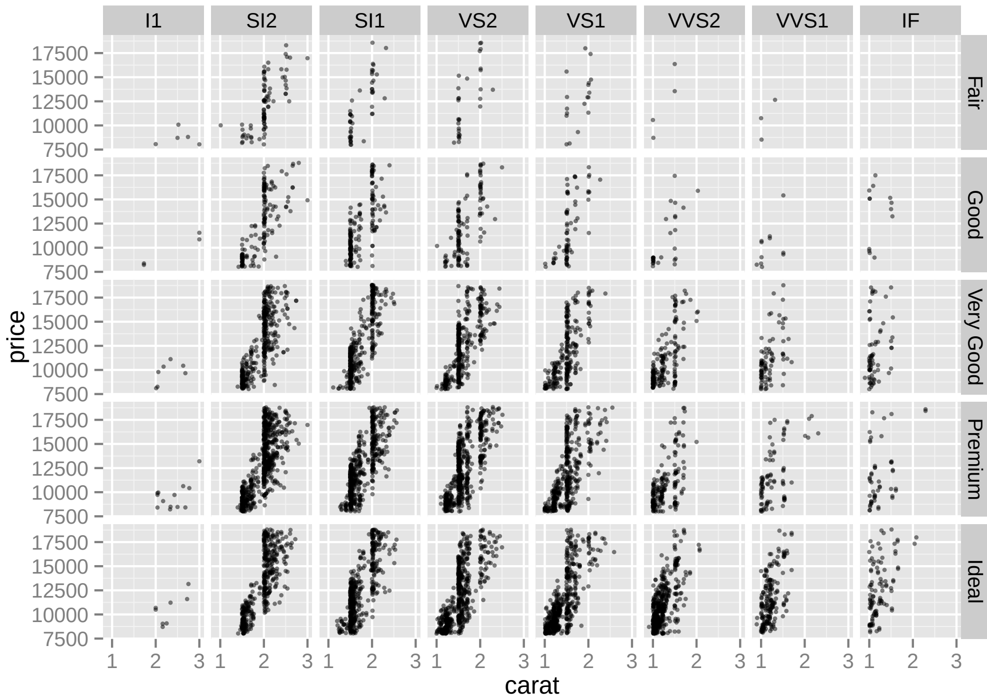
Well, that's obviously not very useful. I want to be able to look at all of my variables at once, but I need them to be on separate plots. This is where the `facet_wrap()` function comes in handy. Our data is in long format, so it is easy to transform this useless plot into something way more useful.

```
ggplot(data = economicsLong, aes(x = date, y = value)) +
    geom_point(size = 1) +
    facet_wrap(~variable, scale = "free_y") +
    labs(x = "Date", y = "")
```



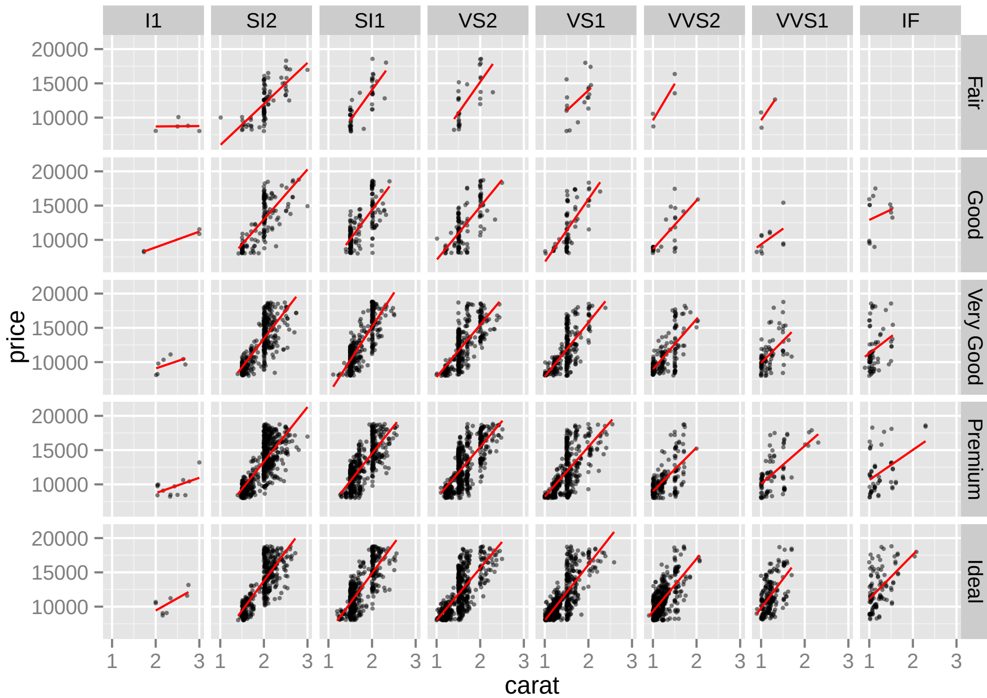
For a better example let's go all the way back to our `diamonds` data set. Say we want to look at the relationship between carat and price, but now we want to look at how that relationship changes for different cut qualities and clarity rankings. Sounds complicated, but it's actually very easy with the `facet_grid()` function.

```
library(scales)
ggplot(data = diamonds[diamonds$price > 8000 &
    diamonds$carat <= 3,], aes(x = carat, y = price)) +
    geom_point(size = 1, alpha = 0.5) +
    facet_grid(cut ~ clarity) +
    scale_x_continuous(breaks = pretty_breaks(n = 3))
```



I had to use the `pretty_breaks()` function in the `scales` library in order to fix the x-axis because the default choice by ggplot took up too much room. I also subsetted the dataset because plotting 50,000 points in a pdf makes it unnecessarily large. What if we want to add a linear regression over every one of these plots? All we have to do is add `geom_smooth()`.

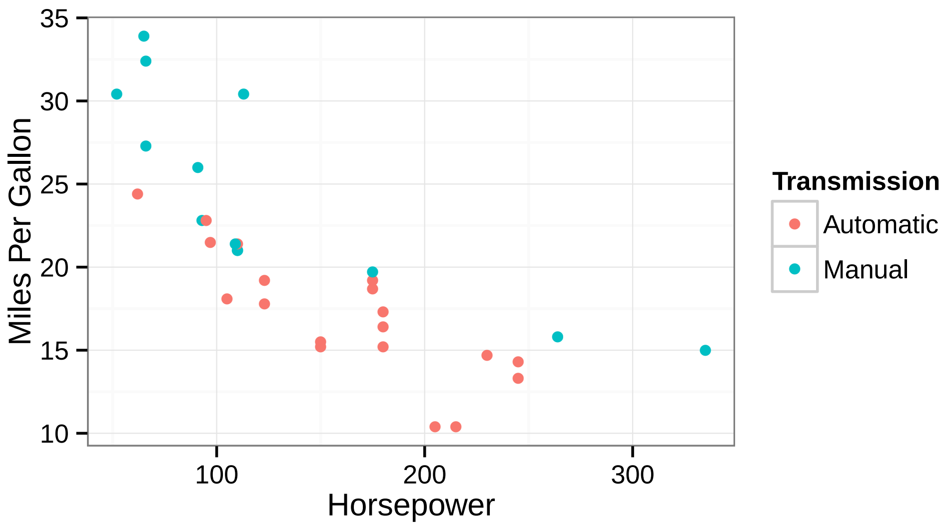
```
library(scales)
ggplot(data = diamonds[diamonds$price > 8000 &
    diamonds$carat <= 3,], aes(x = carat, y = price)) +
    geom_point(size = 1, alpha = 0.5) +
    geom_smooth(method = "lm", se = FALSE, color = "red") +
    facet_grid(cut ~ clarity) +
    scale_x_continuous(breaks = pretty_breaks(n = 3))
```



8 Themes and Colors

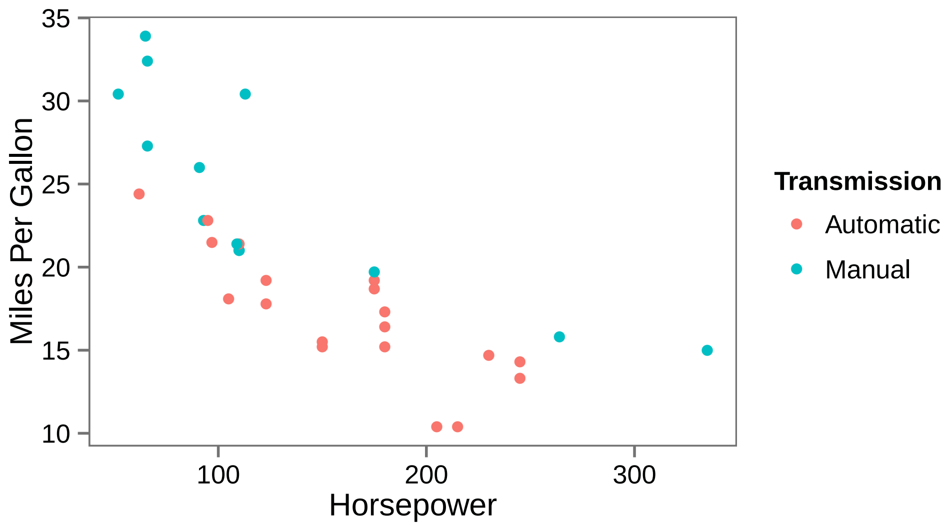
The default theme in ggplot looks pretty nice, but personally I like things to be a bit more minimalist. The `theme()` function allows you to set all sorts of options for how the overall plot looks, but let's just look at the `theme_bw()` function.

```
ggplot(data = mtcars, aes(x = hp, y = mpg,
    color = factor(am))) +
    geom_point() +
    scale_color_discrete(labels = c("Automatic", "Manual")) +
    labs(color = "Transmission", x = "Horsepower",
        y = "Miles Per Gallon") +
    theme_bw()
```



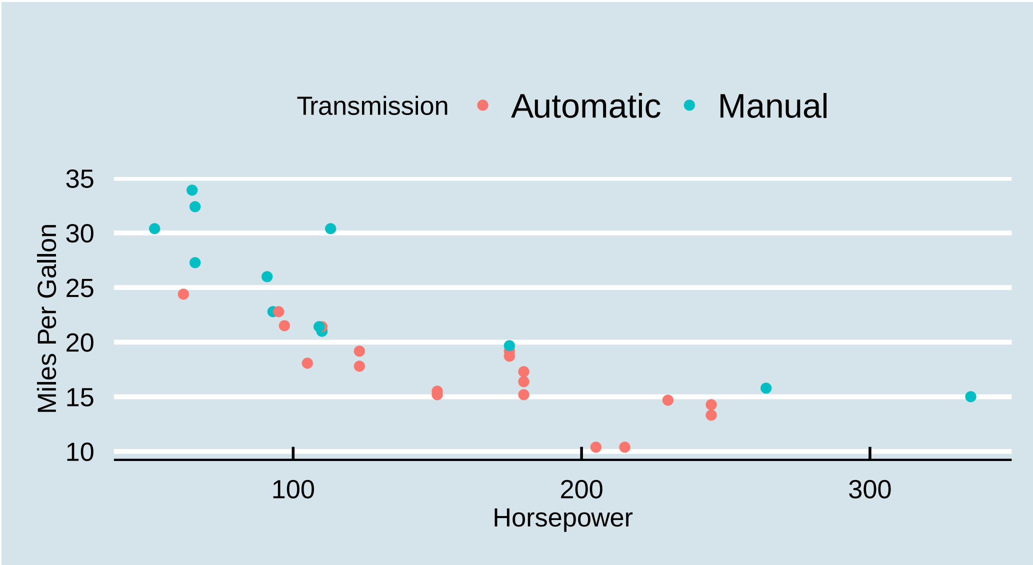
Of course this is mostly personal preference, but I think that looks nicer than the default plot style. The ggplot package comes with many themes already built in, but the `ggthemes` package offers even more. My personal favorite is `theme_few()`. If you're a fan of minimalist plots then this is the theme for you.

```
library(ggthemes)
ggplot(data = mtcars, aes(x = hp, y = mpg,
    color = factor(am))) +
    geom_point() +
    scale_color_discrete(labels = c("Automatic", "Manual")) +
    labs(color = "Transmission", x = "Horsepower",
        y = "Miles Per Gallon") +
    theme_few()
```

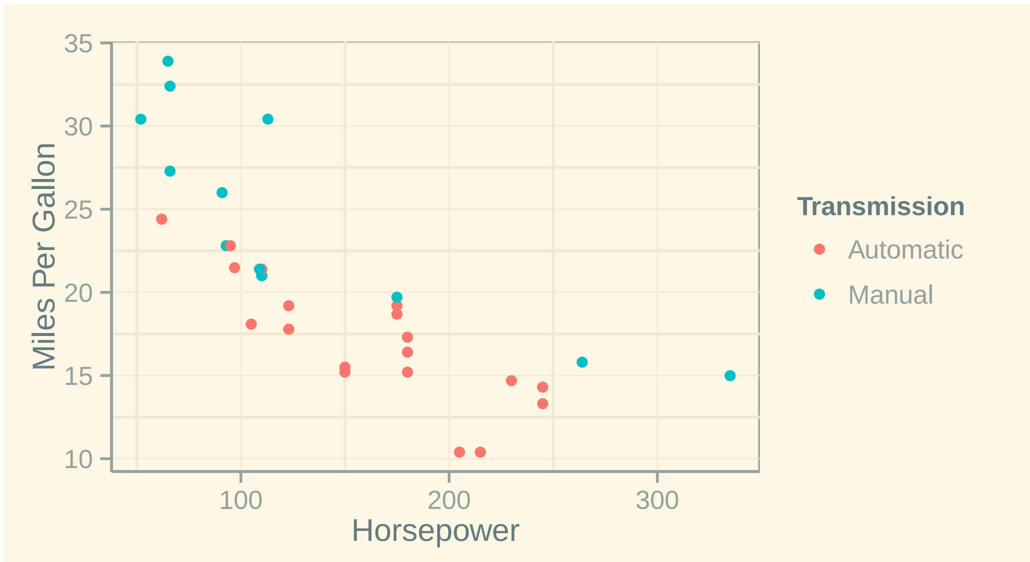


Here are a few more examples:

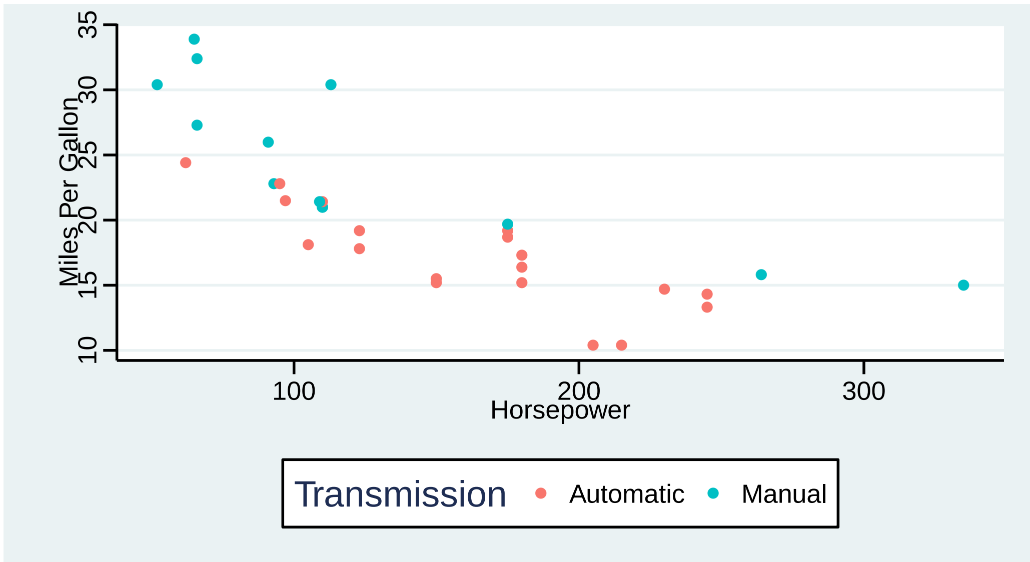
```
ggplot(data = mtcars, aes(x = hp, y = mpg,
    color = factor(am))) +
    geom_point() +
    scale_color_discrete(labels = c("Automatic", "Manual")) +
    labs(color = "Transmission", x = "Horsepower",
        y = "Miles Per Gallon") +
    theme_economist()
```



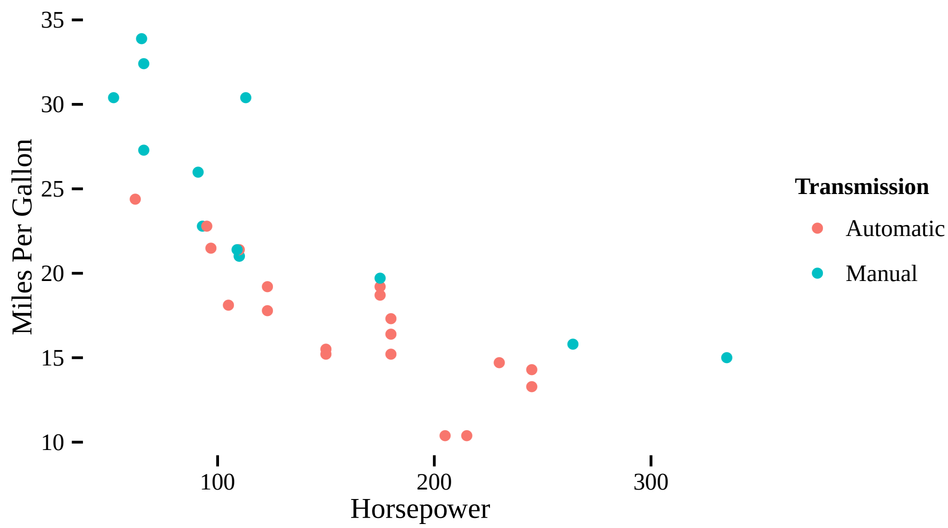

```
ggplot(data = mtcars, aes(x = hp, y = mpg,
    color = factor(am))) +
    geom_point() +
    scale_color_discrete(labels = c("Automatic", "Manual")) +
    labs(color = "Transmission", x = "Horsepower",
        y = "Miles Per Gallon") +
    theme_solarized()
```



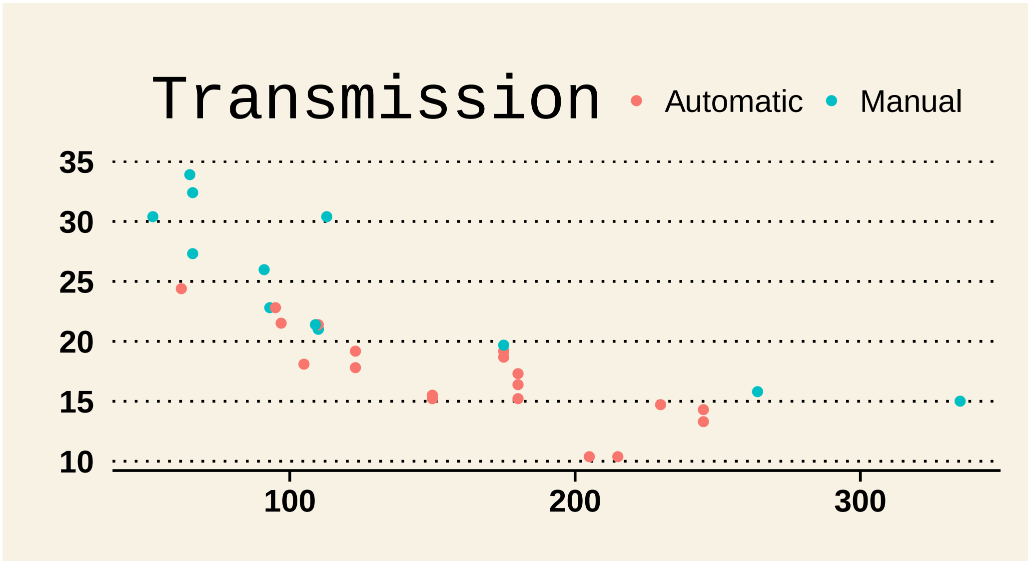

```
ggplot(data = mtcars, aes(x = hp, y = mpg,
    color = factor(am))) +
    geom_point() +
    scale_color_discrete(labels = c("Automatic", "Manual")) +
    labs(color = "Transmission", x = "Horsepower",
        y = "Miles Per Gallon") +
    theme_stata()
```




```r
ggplot(data = mtcars, aes(x = hp, y = mpg,
    color = factor(am))) +
    geom_point() +
    scale_color_discrete(labels = c("Automatic", "Manual")) +
    labs(color = "Transmission", x = "Horsepower",
        y = "Miles Per Gallon") +
    theme_tufte()
```

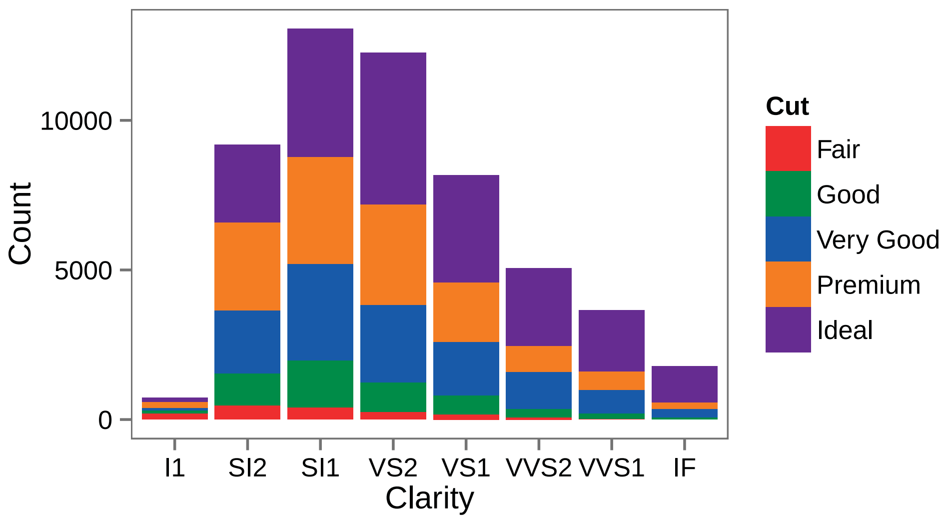



```
ggplot(data = mtcars, aes(x = hp, y = mpg,
    color = factor(am))) +
    geom_point() +
    scale_color_discrete(labels = c("Automatic", "Manual")) +
    labs(color = "Transmission", x = "Horsepower",
        y = "Miles Per Gallon") +
    theme_wsj()
```



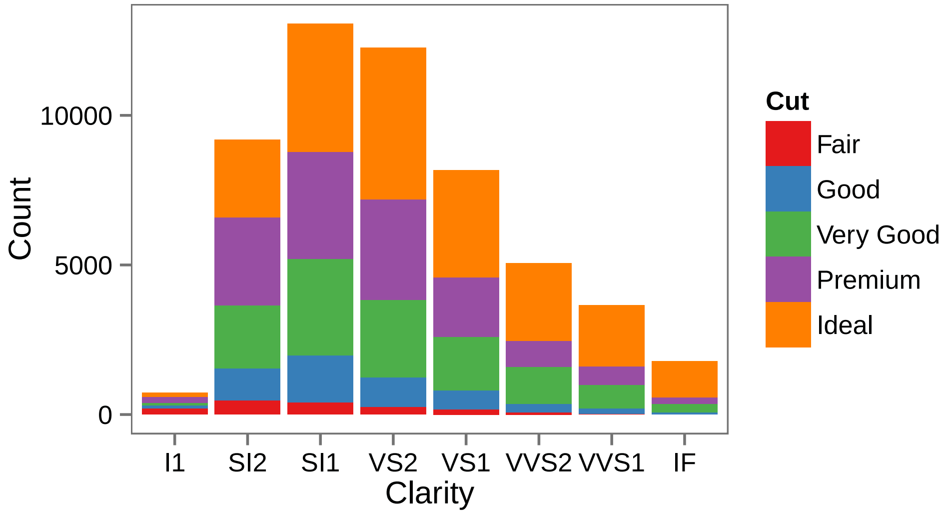
Each one of these themes comes with a color palette that goes along with it. Again, my favorite is the `few` theme, and you can use the corresponding color palette in the following way:

```
ggplot(data = diamonds, aes(x = clarity, fill = cut)) +
    geom_bar() +
    labs(x = "Clarity", y = "Count", fill = "Cut") +
    theme_few() +
    scale_fill_few(palette = "dark")
```

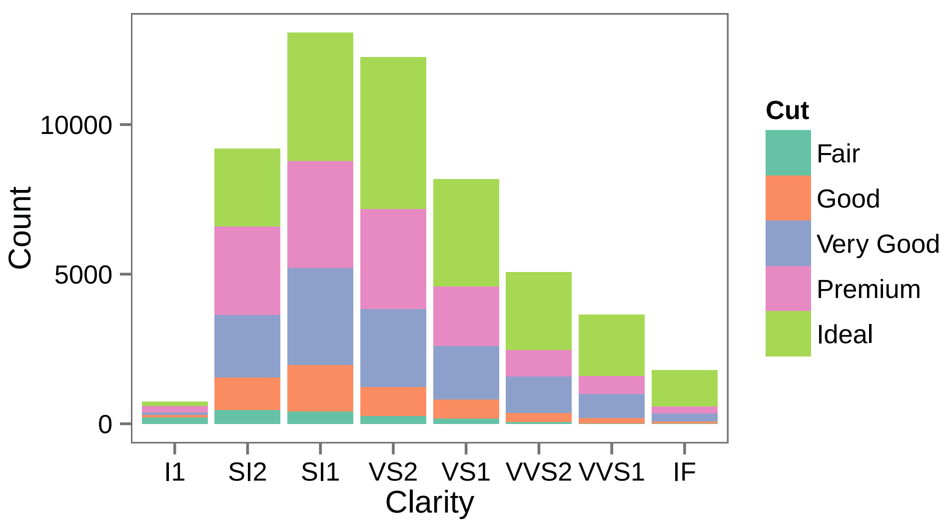


I also feel the need to mention something called Color Brewer (`http://colorbrewer2.org/`). It's made for discrete values on maps but it can be generalized to other types of plots and even continuous values. It offers a lot of really good color palettes, and it comes built into ggplot (the `RColorBrewer` package offers access to these palettes outside of ggplot). Here's an example of a few color palettes.

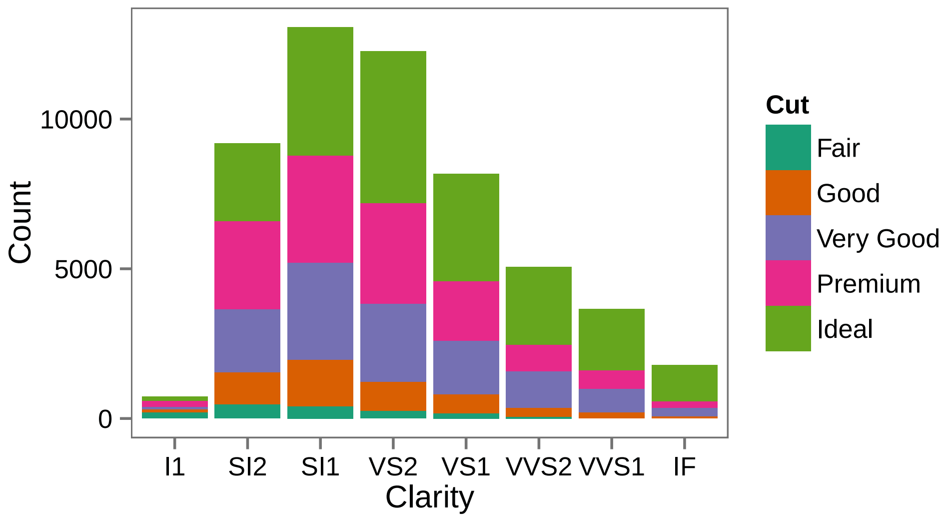
```
ggplot(data = diamonds, aes(x = clarity, fill = cut)) +
    geom_bar() +
    labs(x = "Clarity", y = "Count", fill = "Cut") +
    theme_few() +
    scale_fill_brewer(palette = "Set1")
```




```r
ggplot(data = diamonds, aes(x = clarity, fill = cut)) +
    geom_bar() +
    labs(x = "Clarity", y = "Count", fill = "Cut") +
    theme_few() +
    scale_fill_brewer(palette = "Set2")
```



```
ggplot(data = diamonds, aes(x = clarity, fill = cut)) +
    geom_bar() +
    labs(x = "Clarity", y = "Count", fill = "Cut") +
    theme_few() +
    scale_fill_brewer(palette = "Dark2")
```



What about continuous values? Luckily ggplot has provided `scale_color_distiller()` and similar functions in order to handle this. The following plot colors each point by the average IMDB user rating according to the `Reds` palette from Color Brewer.

```
ggplot(data = movies[movies$budget >= 1000000,],
    aes(x=budget/1000000, y = length, color = rating)) +
    geom_point(size = 1.5, alpha = 0.5) +
    labs(x = "Budget (million USD)", y = "Length (minutes)",
        color = "Rating") +
    theme_few() +
    scale_color_distiller(palette = "Reds") +
    guides(color = guide_legend(override.aes =
```

```
    list(size = 3, alpha = 1), reverse = TRUE))
```